The Remarks Section of this Response is divided into two parts. First, Applicants briefly review what is taught by the Visual Basic reference and what is taught by Applicants' disclosure. This part is provided as an aid to the Examiner to place the two software programs and methods in appropriate context. In addition, Applicants address the substantial differences between the Visual Basic prior art cited by the Examiner and Applicants' invention and indicate why Visual Basic can not and does not anticipate Applicants' invention. Second, Applicants respond to each substantive issue that the Examiner has raised by reference to the Examiner's paragraph number and comments. The Marmelstein rejections are dealt with in response to the appropriate Examiner paragraphs.

AMENDMENT

Please amend the claims as set forth in the attached claim set. The claims presently in the application as well as those amended by this Response are attached starting on separate sheets and identified as required under the revised rules governing claim submission. Please cancel without prejudice claims 12, 19, 39, and 46.

REMARKS

Visual Basic (Gurewich et al.):

Visual Basic is a full featured programming language with a set of logical and arithmetic operators, data structures, and a host of reserved words. Visual Basic utilizes VBX controls, but for a variety of reasons the consensus is that Visual Basic Version 3.0 cited by the Examiner does not treat them as true objects, but rather as a convenient software component type. For that reason Visual Basic is not really considered[1,2,3] to be an object

[1]See for example: Byte Magazine, May 1994, John Udell. Real objects, as OOP (object-oriented programming)

oriented programming system, but rather it is called an event driven programming system that uses components.

VBX controls may be written by a wide variety of groups and individuals but each control conforms to a known standard. Once written, the controls have only the properties and events that the original authors included; that is, the native properties and events. When such controls are used within applications written in C++ and Visual Basic, the controls only present the native properties and events included by the developers of those controls.

Visual Basic has a development interface that allows programmers to arrange the controls on a form, a control browser that allows the developer to preset properties in the VBX control, and a window for the writing of source code in a textural form to operate those controls. However, Visual Basic has no graphical flowchart view for programming. Visual Basic is expressed in a textural "code" and that "code" is compiled. In VB 3.0 (the cited version of Visual Basic) the "code" is compiled into a non-machine language pseudo-code (p-code) before it is interpreted. P-code is not a real machine language, but rather a tokenized version of the textural code where the links have been resolved, the variables have been assigned, and strings have been removed by translation to numeric values. The compiled pseudo code (p-code) is read by the interpreter which subsequently performs the machine code

---

experts rightly point out, rest on the tripod of inheritance, polymorphism, and encapsulation, while VBXes stand only on the single leg of encapsulation.

[2]See for example: Byte Magazine, March 1994, John Udell. A VBX is, after all, only an odd sort of DLL that co-opts the VB message pump in a useful way. (VB, in turn, co-opts the Windows message pump in a useful way.) While a VBX encapsulates methods and data, it isn't inheritable or polymorphic.

[3]See for example: Dr. Andreas Polze, University of Potsdam. Components are rather on the level of classes than of objects. The ultimate Difference, components capture the static nature of a software fragment. Objects capture its dynamic nature.

functions of the program. All the controls on a Visual Basic form are instantiated when the form is opened, and this is the reason that the only way to remove a VBX control from memory is to close the form window. Visual Basic looks in the current directory, the Windows/system directory or relies on the MS operating system "path" command to find VBX controls. If they are not found, the program fails.

In Visual Basic the p-code is contained within a machine language executable program (EXE file). This executable program is linked to an additional associated data linked library (DLL) called VBRUN300.DLL. According to Microsoft:

> "...This run-time DLL provides a number of services for your compiled program, such as startup and shutdown code for your application, functionality for forms and intrinsic controls, and run-time functions such as Format and CLng." (Microsoft Knowledge Base Article - 229415).

What is also in this data linked library (DLL) is the actual run time interpreter that interprets the p-code. In Visual Basic the interpreter is a data linked library (DLL) called from the executable. A data linked library (DLL) is dependent on the executable for its execution and acts more like an overlay file than an independent application. This is the architecture of Visual Basic.

Applicants' Invention:

Applicants believe that the clearest way to distinguish their invention from Visual Basic is to point out the basic differences of their invention by contrasting them to the architecture of Visual Basic outlined above. As a start, it is important to realize that Applicants' invention has neither logical, nor arithmetic operators and as such was unique in the universe of scripting

systems and programming environments at the time of its invention. In Applicants' invention there are only a few reserved words for functions: @Get, @Set, @Killtag, @Additem, @Removeitem, @Refresh and @Setfocus. (refer to VB_FUNCT.CPP line 76) That is the totality of the syntax. There is only one data structure, and that is the ability to store strings in a list of global properties for the application under the object name, "_This", which refers to the Applicants' invention's container.

There is no source code written in Applicants' invention. Hence, there is no compilation of code required and there is no free form textural code sheet for a programmer to use in writing source code to operate the controls. Rather there is a script. Applicants' script contains information that the interpreter can use to find and load the objects required by the script. This novel and unique "smart loading" feature allows Applicants' invention to find and install objects as they are needed when a script is run.

The scripts of Applicants' invention are not compiled into a different form such as p-code. Rather the script is used in a text system that is interpreted one line at a time. When a script is saved, references to other controls in the script are resolved to insure that the expected targets of branching exist and that the names of the variables exist and are unique. There is no tokenizing and no linking of the script to the interpreter. As such, the script is separate from the interpreter and can be maintained and transmitted (such as over the internet) in complete separation from the interpreter. In a typical case, the script can be sent to another locale and used with a run time interpreter and required controls to implement the program at the location remote from the development site. This contrasts with Visual Basic where the interpreter is linked (DLL) to the program executable. The final saved form of Applicants' script is a

compressed version of the text script using the "gzip" compression technique. Compressing the script is done to save disk space and to make transmission of the script over networks more efficient. Aside from the act of compressing the script, the information stays in string form. That is why Applicants refer to it as a script instead of a program.

Applicants' run time interpreter is an independent executable program that reads in a script file and dynamically utilizes the VBX controls referenced in that script. The reason this point is emphasized here is that many of the claims included in the application speak of a script that is maintained separately from the interpreter because the invention scripts are so maintained. It can not be said that the Visual Basic interpreter is separate from the data coded into the Visual Basic program when it is clear that the interpreter is dependent on the executable version of the script; remember, the interpreter is called from the compiled executable. Applicants' invention's interpreter is a free standing application, and executes well without a script. When it has no script to process, it simply stops without error.

The difference between the way Applicants' use a script and an independent runtime interpreter is one of the central differences between Applicants' invention and Visual Basic. As Applicants' interpreter reads the script line by line, it learns which control is to be used and what values to set into that control. This requires the system to link to VBX controls dynamically in response to information in a script. This functional ability is non-trivial and central to the invention.

Another important difference between Applicants' invention and Visual Basic lies in the different ways the two programs handle the VBX controls. Applicants' invention handles controls in a very different way than the techniques used in Visual Basic described above.

Applicants' invention uses the controls as objects because Applicants' invention has one internal object that subclasses the controls on demand and treats them as objects (Applicants refer to this as wrapping). This internal object is called "m_pcvbControlTemplate", or the "template", referred to in source code file VB_REGIS.CPP in lines 183, 185, 186, 189, 191, 256, 376, 394, 410, 426, 851, 868, 882, 910, 917, 975, 994, 1013 and 1030. The use of the internal object makes Applicants' invention a true object oriented system, and not just a component based system. While a developer using Applicants' program can not readily subclass controls [except as provided in a small way by the subroutine VBX (VB_SUB.VBX)], and the controls within Applicants' invention do not suddenly exhibit polymorphism and inheritance to a user of the system [except as provided in a small way by the subroutine VBX], the internal structure of Applicants' invention is such that the system itself treats VBX controls as objects.

One clear result of this wrapping is the treatment of each control within Applicants' invention as if it has properties and events not native to the control itself. A listing of these added properties and events can be found in VB_EXTRA.H. Visual Basic, as noted above, is limited to the properties and events which are native to the control. Visual Basic can not add additional properties to the controls. This difference can not be overstated because, as can be appreciated, the software architecture of Visual Basic is completely different from that of Applicants' invention. The similarity between the two programs is limited to the use of the VBX controls.

Once an object is included in the system of Applicants' invention, the system wraps that object with additional properties and events in such a way that it appears that new

properties and events have been added to the control's native properties and events. Since properties and events are internal to controls, Applicants' invention solved the problem of how to add properties and events to the existing native properties and events that are internal to a software control written to a standard.

In order to accomplish this feat of wrapping, Applicants' invention employs a unique object of its own. Applicants' invention has an embedded object which contains the extra properties and events. This object then acts as a kind of envelope for each individual object that is added to the system. Since Applicants' invention is object oriented, it is possible to have many instances of this embedded object. As taught in Applicants' disclosure, when a new VBX control is added to the system, the added control is wrapped in the envelope of a new instance of the internal object. That instance of the internal object does the job of subclassing the added control, and, in doing so, presents as one object that added control with both its native properties and events and with the extended properties and events provided by the wrapper of Applicants' program.

A novel and important distinction between Applicants' invention and the prior art including Visual Basic is that Applicants' invention is the first system that dynamically used standardized objects in a dynamically-linked interpreted environment and creates an instance of a standard object according to the commands in an interpreted script on an individual basis. This is accomplished in Applicants' invention using an intermediate internal object that acts as a wrapper. That intermediate object is permanently linked to the system, but the objects that do the work defined by the Applicants' invention script are not so linked, instead they reside inside the wrapper of the intermediate object. Applicants' invention is generalizable to many

different types of components as taught in the specification:

"While the present implementation of the invention is designed to directly operate standardized Microsoft Visual Basic VBX controls as objects, the manner in which the invention wraps the objects is equally usable with objects written to standards other than the Microsoft VBX control." (Specification: page 7, line 22 - page 8, line 3).

Two further features of Applicants' invention should be mentioned since there are no counterparts in Visual Basic. First, Applicants' invention has a Time Track system to facilitate the development of timed material, (ref TRK_CLIP.CPP, TRK_DRAW.CPP, TRK_FUNC.CPP, TRK_MOVE.CPP, TRK_PRN.CPP, AND TRK_SELE.CPP). Second, Applicants' program has a Flowchart view for programming, (ref MAPCLIP.CPP, MAPMOUSE.CPP, MAPNODE.CPP, MAPNODE.H, MAPPATH.CPP, MAPPERVI.CPP, MAPPERVI.H, MAPPROPF.CPP, MAPPROPF.H, MAPSCROL.CPP).

In summary, Visual Basic 3.0 can not and does not anticipate Applicants' invention for at least the following reasons:

1) a compiled program (VB) containing coded instructions which executable calls an interpreter within a DLL, does not anticipate a non-compiled program that produces a script, that may be separately maintained, for line by line interpretation by a free standing run time interpreter which links VBX controls dynamically in response to information in a script;

2) a program (VB) which is not object oriented and can not add additional properties and events to those native to the controls, does not anticipate a program that is object oriented and uses an internal object to wrap standard controls to add additional

properties and events not native to the standard controls; and

3) a program (VB) which does not have a time track system or a flowchart view for programming does not anticipate a program that does.

Other significant differences between Visual Basic and Applicants' invention will be highlighted in response to the Examiner's remarks. However, all of Applicants' responses with respect to why Visual Basic does not anticipate Applicants' invention rest on one underlying and immutable fact; namely that the claims in this application are to a system of use of wrapped objects in interpreted environments, something not taught anywhere in the prior art. Applicants' arguments presented above are hereby incorporated into Applicants' responses set out below.

It is clearly evident from page 580, cited by the Examiner, that Visual Basic is used to make executable programs. Note that the WEST application is there referred to as an executable program, WEST.EXE. Applicants' invention, on the other hand is a system "in which both the objects and the script may be maintained separately". In the claims presented in the present application, it is consistently repeated that the scripts are maintained separately and independently which is only possible in interpreted systems that use a script, and such a statement is impossible to make about systems like Visual Basic and C++ applications where code is compiled into executable programs. Visual Basic and C++ treat objects in a fundamentally different manner than Applicants' invention does. The only similarity is the type of software component. The treatment of components within Visual Basic and C++ do not make obvious the very different treatment of objects in Applicants' invention. Any reference to Visual Basic and other compiled languages such as ADA, Visual Basic, and C++ is made

immaterial by this fact.

Structure of Examiner's Rejections and Applicants' Response:

Many of the method steps are identical across many of Applicants' claims although they may be combined with other steps in some claims. The Examiner has sensibly dealt with this situation by grouping his claim rejections so that claims having substantially identical method steps are placed together. However, for the claims which have a step in common with another claim but also have additional steps, the Examiner has separately repeated his rejection of the common steps and added a rejection to the additional step. The Examiner has also separately repeated a rejection where a claim is in dependent form. For example, while claims 1, 28, 55, 56,and 57 are grouped together, claims 23, 25, 26, 27, 50, 52, 53, and 54 having at least one step in common with claims 1, 28, 55, 56, and 57 are treated separately. Thus, as one reads through the office action, one comes upon the identical rejection to the same method step time and time again.

Applicants believe that responding to each rejection by the Examiner following the order of the office action is appropriate. However, the repetition of a response to identical rejections in several claims adds nothing but length to the response. Accordingly, Applicants respond to the rejection of each method step the first time that rejection appears. Subsequently, when appropriate in responding to rejections of claims containing additional steps, the initial response is referenced without repetition.

Examiner's Paragraph 6:

The examiner has rejected claims 1, 4-28 and 31-57: "...under 35 U.S.C §102(a) as

being anticipated by Gurewich et al ("Gurewich"), Master Visual Basic 3, Sams Publishing 20, 1994."

Examiner's Comments: **Claims 1,28,55,56 and 57**

Gurewich discloses at least:

a. *means for wrapping objects with additional properties and events beyond those properties and events internal to the object.*(see at least Gurewich, chapter 19, section "attaching Code to the Multitasking Check Box, page 573-576; chapter 20, section "The WEST Program" page 580 et seq., in particular the "Note" at page 584); *and*

b. *means for utilizing the additional properties and events to link and sequence the objects* (see at least chapter 20, section "The WEST Program" page 580 et seq., in particular the "Note" at page 584).

Applicants' Response:

The extension of the property list and event list in Applicants' invention is very different from the activities described in Gurewich. Chapter 19 of Gurewich describes the use of VBX controls within a Visual Basic program. What Gurewich describes is not wrapping of an object with new properties and event, but rather the creation of Visual Basic code segments that are called as the result of a native event occurring within a control. In Gurewich it is explained how an internal event from an existing VBX control can be used to call such code segments.

The cited literature, Gurewich, pages 573-576, and "the West program" pg 580 and 584 describe techniques for using an existing internal event issued by a software component to begin execution of a segment of code written in Visual Basic. Applicants respectfully point out

that contrary to what the Examiner has stated, there is no information contained in these sections that teaches wrapping an existing object with additional properties and events. The cited material merely explains how to use an existing event to trigger the execution of a code segment. Further, Applicants respectfully point out that, contrary to what the Examiner has stated, there is no explanation of a means for using any extended properties or events, rather there is an explanation of a means for using an existing event. Applicants respectfully submit that the Examiner has misinterpreted the art and apparently believes that, just because Visual Basic utilized VBX controls, it must necessarily anticipate Applicants' invention. As has been pointed out above, this is simply not the case. The only similarity that exists is due to the common use of VBX controls by both programs. The manner in which the controls are utilized is very different in both systems.

In Applicants' invention, the VBX controls are not instantiated as a group when a form is opened as in Visual Basic, but rather they are individually instantiated upon demand based on the content of the script. These objects are also treated as objects as they are subclassed and contained in an intermediate object which acts as a wrapper for the control, and that presents the control to the system in an extended form so the objects can perform in ways not envisioned by the developers of the components that have become objects within Applicants' invention. (Use in Applicants' invention of these added events can be seen in VB_NEXT.CPP in lines 205, 207, 216, 368, 370, etc.). In Visual Basic the components have only their internal (or native) properties and events. These native properties and events are used by the interpreter, according to the source code compiled in p-code, to command the components in ways anticipated by the developers of those components.

By way of further answer, a close examination of the material cited by the Examiner further clarifies the distinction between Applicants' extension of the properties and events which can be associated with each object and Visual Basic's limitation to using the native properties and events hard coded into the controls. Gurewich describes the use of the "Click" event to trigger the execution of a subroutine (see page 573). The subroutine mentioned by Gurewich is external to the object. It is a code module written in Visual Basic and does not add capabilities to the control; rather it uses the existing internal properties of a variety of controls. It is true that Gurewich uses the verb "attach", in relation to the connection of this subroutine with the "Click" event. The form of "attachment" is calling a separate subroutine, but nowhere in the cited literature, or elsewhere in the art of Visual Basic programming is there any mention of wrapping objects with additional properties or events in such a way that those properties and events act as extensions of the internal properties and events.

The "Note" cited by the Examiner describes the use of a Visual Basic form (window) to hold a number of independent VBX controls. It shows their settings as documented in a tabular form as discussed earlier, and it details the use of a midi file and a control array. [A control array is a Visual Basic data structure that is internal to the system.] Creating an array of existing controls does not add properties or events to those controls, it only allows them to be put in memory and referenced by an index. The placement of a control on a form in such a way that that control is surrounded by other controls does not wrap the first control with new properties and events, it just puts it on a form in an unchanged manner. The use of computer code as described in the West program example to operate these objects is not an object oriented program wrapping of the objects with new properties and events in the way done by

the subclassing of Applicants' invention, rather it is an example of using existing properties and events in the normal manner.

Visual Basic does not have additional properties beyond those hard coded in the VBX controls. It cannot, therefore, anticipate a system in which additional properties are implemented by a wrapper object and utilized to link and sequence objects. This difference alone is sufficient to overcome the Examiner's basis for rejection. Further, however, the Examiner should understand that Applicants' invention uses the additional properties and events implemented in the wrapper to link and sequence the objects one at a time. See discussion concerning how the additional properties and events are utilized in Applicants' response to the Examiner's rejection of claims 6-7 and 33-34.

Examiner's Comments: **Claims 4 and 31**

Gurewich discloses at least:

a. *a development environment and an interpreting run time environment* (see at least chapter 14); *and*

Applicants' Response:

The cited literature, Gurewich, chapter 14, does include references to a development state and a runtime state. Careful examination of Capt 14 also clearly shows that the runtime state is the execution of an executable application written in Visual Basic, and at no time is there any indication that the Visual Basic system has a separate independent runtime interpreter that is independent from the compiled p-code. In fact the opposite is clearly made evident on page 471 when Gurewich states (in reference to the storage of data):

"...That is, after you make an EXE file of your application, the BMP file is an integral part of the EXE file."

Those familiar with programming understand that EXE files are executable programs. While applications are being developed in the Visual Basic environment it is possible to execute them in a preview mode. Gurewich repeatedly alludes to the method for doing this. One such example is on page 463:

"...To see your code in action, Select Save Project from the File Menu. Select Start from the Run Menu"

The series of steps indicates that the code must first be saved, then it is quickly compiled and run on command from within the development environment. Such technology has been common in development systems since the 1980's. An example of this technology can be found in the program Turbo Pascal by Borland International. By way of contrast, the architecture of Applicants' invention permits a preview execution feature wherein the existing script is interpreted from memory, thus allowing for a variety of execution options. One such option is a "run from" option that allows the developer to start the interpretation of the script from any point in the script. This is not possible in Visual Basic 3.0 with the compiled Visual Basic system.

Since both claims 4 and 31 refer to an interpreted system using a script, it is clear that Applicants' inventive system and Visual Basic are dissimilar since a system based on compiled code can not make obvious a system based on the completely different technology of a script used by an separate independent interpreter.

[It is worth noting as an aside, that sometime around 1997 (well after Applicants'

priority date of 1994) Microsoft developed a scripting system called Visual Basic Script, VBScript. Microsoft's scripting language was a subset of the original Visual Basic system, and maintained the same source code syntax as Visual Basic, but a smaller feature set. The salient point here is that the VBScript "code" is interpreted line by line and not compiled, showing that Microsoft itself appreciated and recognized the difference between the two methods.]

Examiner's Comments: **(Claims 4 and 31 continued)**

b. *means for utilizing objects, by specifying property values according to the script*(see at least chapter 14).

Applicants' Response:

Visual Basic uses VBX controls and specifies property values, but it does so in a very different way than Applicants' invention. Visual Basic uses compiled source code (not to be confused with Applicants' script) that is attached to a form window. That code tells the system how to make the (see above) VBX controls on that form window behave. In Applicants' invention, the script is interpreted as text on a line-by-line basis. Since Applicants' script is interpreted line by line, it works on a dynamic object by object basis. Since Visual Basic is interpreted from compiled p-code and acts on a form basis and the controls are instantiated as a group, it is clear that the methods used in Visual Basic and in Applicants' invention are so very different as to make it impossible to say Visual Basic makes Applicants' invention obvious.

Gurewich, chapter 14, pages 463 and 471, clearly states that Visual Basic does not use a script that is interpreted line by line as Applicants' invention's script is. Rather Visual Basic uses computer code that is compiled and saved in an EXE file. Since Visual Basic is a

compiled system that uses a standard programming language, it is not a scripting system which is interpreted line by line. Even Microsoft recognizes this as noted earlier. Visual Basic is not at all an object oriented software system, rather it is a component based software system. A component based system can not make obvious a system like Applicants' invention which is internally object oriented.

Visual Basic is compiled into p-code and that p-code resides within an executable module. The executable then links to a DLL to run an interpreter. There is no listing of properties and events for a separately maintained interpreter to read. A thing can not be both linked and separate at the same time. Therefore, it can not be said, as the Examiner suggests, that Visual Basic teaches "...a means for utilizing objects by specifying property values according to the script" because Visual Basic does not use a script. In Applicants' invention, the script is a separate file that is loaded into the interpreter and there the script's contents are utilized on a line by line basis as the interpreter reads the script.

Throughout Gurewich, chapter 14, there are code examples wherein VBX properties are made equal to explicit values and to values in other controls (see page 472). Visual Basic directly modifies the contents of the properties of each VBX control. In contrast, the interpreter of Applicants' invention proceeds to read through the script and, when necessary, dynamically loads a VBX control into an internal ("TEMPLATE") object. Subsequently, the interpreter transfers property values stored in the script into the new instance of the object (combination of the VBX control and its "Template" wrapper) before the object is instantiated. When appropriate, property values from other objects are also transferred into the new object.

The means by which the two systems use VBX controls are so different that it is clear

Visual Basic neither anticipates nor makes obvious Applicants' invention. The way Visual Basic uses "code" and Applicants' invention uses "script" is significantly different, and the use of compiled code in Visual Basic does not make obvious the script technique employed in Applicants' invention. As mentioned above, the means for specifying property values in Visual Basic and in Applicants' invention as the application is running are significantly different. Visual Basic moves information directly into the properties of VBX controls. In Applicants' invention, information is moved in an entirely different manner. Applicants' invention uses a wrapper (template) object which subclasses the VBX control. The script directs the instantiation of the template object and its dynamically loaded associated sublclassed VBX control and provides the required information to the template object which then communicates the information to the VBX control.

Examiner's Comments: **Claims 5 and 32**

The rejection of the base claim is incorporated. Gurewich further discloses a means for communicating among objects through the exchange of property values (see at least Figure 14.13; Figure 14.26 and related discussion in the specification)

Applicants' Response:

Applicants submit that they have overcome all bases for rejection of the base claim and that this dependent claim can not be obvious when the base claim is not obvious. Gurewich, Figs 14.13, 14.26, and related discussion do show that Visual Basic can be used to transfer information from one VBX control to another. That is a well known function in Visual Basic and other compiled development systems like C++ and ADA. However, Applicants submit

that the reference does not make obvious the means by which the present invention accomplishes the act of intercommunication in a system that interprets the script line by line as is found in claim 4. Applicants do not claim inventing communication between objects in all computer systems. Rather claim 5 is a dependent claim of claim 4. Claim 4 teaches the art of interpreted object oriented technology. Claim 4 is neither made obvious by software that is compiled nor is it made obvious by software that is non-object oriented like Visual Basic.

The means for utilizing the objects according to the script is the software invention that wraps each VBX component inside an object internal to the interpreter and development system. (As noted above, the internal object is referenced in the code attachment as "template object.") This internal object implements added properties and events which are not native (internal) to the VBX controls. It also is the means for using the standard VBX component in a dynamic way. Using an internal object to dynamically wrap a standard VBX component before using that control's property values is very different from the way Visual Basic directly controls VBX controls in a form based system. The way information is moved in an interpreted object oriented system (as claim 5 states) is similarly not made obvious by the referenced art.

Applicants submit that the use of a number of copies of the internal object (as detailed above) each of which subclasses a VBX control (as the term "subclassing" is understood in the art of object oriented programming) and implements additional properties and events, requires that the movement of information between the internal objects and ultimately down to the VBX controls, is significantly different than the technique used in Visual Basic 3.0 to move data directly between VBX controls. [Applicants' invention moves property values through intermediate (template) objects wherein some of the values are used by the template and some

of the values are passed to the VBX control. Visual Basic moves property values directly to the controls.] The functional difference, as explained here in detail, is so non-trivial that it can not be legitimately concluded that Visual Basic renders obvious a system like Applicants' invention just because both kinds of systems employ VBX controls.

As discussed above, the means of communication taught by Gurewich relates to Visual Basic, and Visual Basic is not an object oriented language. Applicants' invention is an object oriented language that wraps objects in a novel manner not anticipated by Gurewich. While VBX controls are used in both Visual Basic and Applicants' invention, the means of such use are quite different. Similarity of effect does not mean that the means employed in Visual Basic anticipate the totally different means employed in Applicants' invention.

Examiner's Comments: **Claims 6-7 and 33-34**

The rejection of the base claim is incorporated. Gurewich further discloses a means for communicating among objects wherein an event generated by an object triggers an instance of another object (see at least Figures 14.26 and related discussion in the specification).

Applicants' Response:

Applicants submit that they have overcome all bases for rejection of the base claim and that this dependent claim can not be obvious when the base claim is not obvious. Visual Basic is form based. The VBX controls are utilized within form windows, and, when the window is opened, all the controls on that window are automatically instantiated. (It is not necessary for a Visual Basic form to contain a VBX control, but they most often do.) An event triggered by a VBX control in a form may be used by the Visual Basic program to begin a series of commands that effect other already instantiated controls within that form. Such an event may

be used to trigger the instance of a form window, but it does not directly trigger the instance of another control. In contrast to this, in Applicants' invention an event in one control always triggers the instantiation of another control with the single exception of the control that is used to end the program.

By way of further answer, the "drag and drop" example found in the reference does not indicate an example of communication between objects, rather it explains how an event within a VBX control can trigger the execution of a subroutine. Note, in the cited reference, the routine is called a "subroutine" and not an "object." Shown below is the subroutine written using the Visual Basic syntax as found the middle of page 478:

"Sub Form_DragDrop (source as Control,

X as Single,

Y as Single)

Source.Move X,Y

End sub"

This is clearly not an example of the use of an object.

Applicants appreciate the Examiner's confusion, which has led to this rejection, that was most probably generated by Microsoft's unfortunate use of the term "object" in the text. The point of confusion can be found in the use of the term "object" as it relates to "forms" within Visual Basic version 3.0. It is true that the various "windows/pages" visible to the user of a Visual Basic program are referred to as "forms", and it is also true that those "forms" are called "objects" for the purposes of explaining that system. What is not necessarily clear to one not schooled in the object oriented programming art is that that nomenclature has nothing

to do with the generally accepted use of the term "object" as it is known to those familiar with the art of object oriented programming.

A Visual Basic "form" of the kind disclosed in the reference does not behave like an object as understood in the object oriented programming art; rather it is a procedure that Microsoft for reasons unknown decided to call an "object". As taught in the Visual Basic reference, there was a minimal set of parameters that were inherent to the procedure, and the programmer could use these parameters as required. A more generally accepted understanding of the "forms" found in Visual Basic version 3.0 is that "forms" are windows, and controls are components. It is reasonable for programmers to pass information into those windows (forms) via parameters as one would pass variables into a procedure in any structured language because these forms hold the pieces of code that control the components and the only way to use the system is to pass parameters between these code segments.

While Microsoft called these forms/windows/procedures "objects", such nomenclature did not make them true objects as understood by those familiar with the art of object oriented programming. There was no inheritance to those forms, and there was no polymorphism to them as found in a true object oriented programming object. Similarity of name does not establish identity of function.

In addition to this distinction between object oriented programming objects and things that Microsoft called objects, it is useful to remember that Applicants' invention operates in a completely different manner. In Applicants' invention, the objects communicate between themselves with a small bit of help from the Applicants' container. When an event is triggered by an object, the run time interpreter traps that message and matches the name of the target

object with the name of the object associated with the event held within a table in the wrapper which the wrapper contains for this purpose. Once the look-up is accomplished, the target object is instantiated within a new internal object wrapper for that new object according to the reference to that object found in the script. In Applicants' invention there is no code base called within a procedure as in Visual Basic. For all of the foregoing reasons, Visual Basic does not anticipate Applicants' invention.

Examiner's Comments: **Claims 8 and 35**

Gurewich discloses at least:

a. *a development environment and an interpreting run time environment that have no logical or arithmetic operators* (see at least chapter 14); *and*

Applicants' Response:

Applicants incorporate their response with respect to claims 4 and 31 above. Gurewich, chapter 14, page 465, clearly indicates that Visual Basic has the logical operator known as a "FORNEXT" loop. Since Visual Basic has this logical operator, it can not be said to make obvious a system such as Applicants' that does not have a logical operator. It should also be noted that syntax for use for the logical operator "FORNEXT" requires addition of a counter. This clearly shows that Visual Basic has at least one arithmetic operator, addition. Since Visual Basic has this arithmetic operator it can not be said to make obvious a system, such as Applicants', that does not.

Examiner's Comments: **(Claims 8 and 35 continued)**

b. *a means for utilizing objects by specifying property values according to the script* (see at least chapter 14).

Applicants' Response: Applicants incorporate their response with respect to claims 4 and 31 above.

Examiner's Comments: **Claims 9 and 36**

The rejection of the base claim is incorporated. Gurewich further discloses *a means for communicating among objects through the exchange of property values* (see at least Figure 14.13; Figure 14.26 and related discussion in the specification).

Applicants' Response:

Applicants submit that they have overcome all bases for rejection of the base claim and that this dependent claim can not be obvious when the base claim is not obvious. Further, Applicants incorporate their response with respect to claims 5 and 32 above.

Examiner's Comments: **Claims 10-11 and 37-38**

The rejection of the base claim is incorporated. Gurewich further discloses *a means for communicating among objects wherein an event generated by an object triggers an instance of another object* (see at least Figure 14.26 and related discussion in the specification).

Applicants' Response:

Applicants submit that they have overcome all bases for rejection of the base claim and that this dependent claim can not be obvious when the base claim is not obvious. Further,

Applicants incorporate their response with respect to claims 6 - 7 and 33 - 34 above.

Examiner's Comments: **Claims 12 and 39**

Gurewich discloses at least:

a. *a development environment and an interpreting run time environment that have not definable data structure architecture* (see at least chapter 14); *and*

Applicants' Response:

Applicants have closely examined all the claims and have determined that these claims were too broadly written and would encompass the single facility to store strings found in Applicants' invention. Consequently, Applicants have canceled these claims.

Examiner's Comments: **(claims 12 and 39 continued)**

b. *means for utilizing objects by specifying property values according to the script* (see at least chapter 14)

Applicants' Response:

Applicants incorporate their response to claims 12a and 39a.

Examiner's Comments: **Claims 13 and 40**

The rejection of the base claim is incorporated. Gurewich further discloses *a means for communicating among objects through the exchange of property values* (See at least Figure 14.13; Figure 14.26 and related discussion in the specification).

Applicants' Response:

Applicants submit that they have overcome all bases for rejection of the base claim and that this dependent claim can not be obvious when the base claim is not obvious. Further, Applicants incorporate their response with respect to claims 5 and 32.

Examiner's Comments: **Claim 14-15 and 41-42**

The rejection of the base claim is incorporated. Gurewich further discloses *a means for communicating among objects wherein an event generated by an object triggers an instance of another object* (see at least Figure 14.26 and related discussion in the specification).

Applicants' Response:

Applicants submit that they have overcome all bases for rejection of the base claim and that this dependent claim can not be obvious when the base claim is not obvious. Further, Applicants incorporate their response to claims 6 - 7 and 33 - 34 above.

Examiner's Comments: **Claims 16 and 43**

The rejection of the base claim is incorporated. Gurewich further discloses *a means for adding additional programming constructs by employing objects that perform the function of programming constructs wherein unlimited expansion of programming capabilities is achieved* (see at least "Writing the code of the FormLoad() procedure", p. 585 et seq.; "Attaching the code to cmdPlay button", p. 591 et seq.).

Applicants' Response:

Applicants submit that they have overcome all bases for rejection of the base claim and

that this dependent claim can not be obvious when the base claim is not obvious. By way of further answer, Applicants note that Visual Basic has its own internal programming constructs, e.g. FORNEXT, while the Applicants' invention has none. It hypothetically possible that programming constructs created as objects written to the specification of a VBX control could be added to Visual Basic because such a control would adhere to the standard. However, doing so would add no little utility since such constructs are already embedded in the Visual Basic system and Applicants are unaware of any such use. Such objects would be totally redundant in Visual Basic. In fact, no one, except Applicants, has created these constructs because it was probably recognized that they were unnecessary in Visual Basic. To demonstrate that the use of additional objects as programming constructs is novel, it is only necessary to note that since 1994, the original filing year for Applicants' invention, there have been no such objects offered in the market except those offered through the company operated by the inventors. Accordingly, Visual Basic does not anticipate Applicants' invention.

Examiners Comments **Claims 17 and 44**

Gurewich discloses at least:

a. *a run time program* (see at least chapter 14); and

Applicants' Response:

Applicants' incorporate and repeat their response to the Examiner's comments on claims 4 and 31 above. The response above demonstrates that Gurewich, and Visual Basic, neither anticipate nor explain an interpreted system like the Applicants' invention. To briefly repeat the reasons listed above: Visual Basic is not an object oriented system, and Visual

Basic's interpreter contained in a DLL (VBRUN300.DLL) is not separate from a script as it is delivered for execution. It is impossible to be linked (DLL) and dependent (as the interpreter is in Visual Basic) and at the same time "separate" as claimed for Applicants' script.

In Applicants' interpreted system, the program interpreter is referred to as the runtime program. This interpreter, or runtime program, is an independent executable program that reads textural scripts. This program does not change no matter what is in the script. The runtime does not depend on the existence of the script to execute; that is, it is not called from the script or linked to the script through a DLL. It is not called from a compiled EXE. It is independent from the script in its final form. The script is a separate file that is not contained within an executable file, rather it is read by one. For this reason Applicants' claim 17 refers to: *"A computer implemented system employing objects and interpreting a script in which both the objects and the script may be maintained separately."* Applicants respectfully submit, that since Visual Basic, C++, ADA and similar programs do not have scripts that are maintained as separate and independent files from the runtime program, there is no way to conclude that these types of programs, which do not employ Applicants' architecture, can render Applicants' system obvious.

An additional benefit of Applicants' system architecture is that it results in smaller and more easily and readily modifiable programs. The architecture of standard applications like those made with C++ and Visual Basic is such that large programs are the result. If one wants to update such a program, the old program is removed and a new one is installed. The size of each of these new installations can easily be many megabytes. Applicants' architecture allows for applications to be easily updated. Instead of removing all the old executable code, in

Applicants' invention a small script is substituted in place of the original script. In fact, only a portion of the original script may need to be replaced. Once the runtime system is in place, Applicants' scripts are 5-10% the size of comparable C++ or Visual Basic applications. Also there is no need to remove the old executable and reinstall a new one since Applicants' script is not an executable module but rather relies on an existing unchangeable executable program, the runtime interpreter.

Examiner's Comments: **(claims 17 and 44 continued)**

b. *means for utilizing objects according to the script* (see at least chapter 14)

Applicants' Response:

Applicants' incorporate their response with respect to claims 4 and 31 above. The cited Gurewich reference, chapter 14, pages 463 and 471, clearly states that Visual Basic does not use a script, but rather uses computer code that is compiled into executable form and saved in standard EXE files. Visual Basic can not be said to make obvious an object oriented system such as Applicants that uses scripts in the utilization of objects.

Additionally the means by which the objects are utilized within Applicants' invention is by dynamic linking and an interpreted script. The ability to link to and instantiate subclassed Visual Basic controls dynamically during the line by line interpretation of a script is central to Applicants invention. Applicants invented a method to place variables in the control by means of an internal "template" object, to move data between instances of the internal object (each of which holds a different VBX control), to cause the VBX control to perform its intended work while being within the internal object, and to deinstantiate a specific instance of the internal

object upon command, all when the script called for these events to occur.

Examiner's Comments: **Claims 18 and 45**

Gurewich discloses at least:

a. *a run time program that has neither arithmetic nor logical operations* (see at least chapter 14); *and*

Applicant's Response:

Applicants incorporate their response with respect to claims 17a and 44a, and 8 and 35 above.

Examiner's Comments: **(claims 18 and 45 continued)**

b. *means for utilizing objects according to the script* (see at least chapter 14)

Applicants' Response:

Applicants incorporate their response with respect to 17b and 44b, and 4 and 31 above.

Examiner's Comments: **Claims 19 and 46**

Gurewich discloses at least:

a. *a run time program that has no definable data structure architecture* (see at least chapter 14); *and*

Applicants' Response:

Applicants have closely examined all the claims and have determined that these claims were too broadly written and would encompass the single facility to store strings found in

Applicants' invention. Consequently, Applicants have canceled these claim.

Examiner's Comments: **(claims 19 and 46 continued)**

b. *means for utilizing objects according to the script* (see at least chapter 14).

Applicants' Response:

Applicants incorporate their response to claims 19a and 46a.

Examiner's Comments: **Claims 20 and 47**

Gurewich discloses at least:

a. *means for instantiating objects* (see at least chapter 20, section "The visual implementation of the WEST program", p. 581 et seq);

Applicants' Response:

Gurewich, chapter 20, describes how to write "the code that instantiates the controls [objects] and variables of the WEST program inside the formload() procedure." Visual Basic places VBX controls within windows called forms. The Visual Basic interpreter actually instantiates the VBX controls, and there is no indication that this interpreter treats these VBX controls any differently that Microsoft's own C++; that is to say, by using the ordinary techniques of the day. These techniques were thoroughly discussed in the literature of the time and the conclusion was that there was no object orientation to their treatment, rather they were considered like DLL's that supported automated inspection and conformed to a set specification.

Applicants' invention teaches instantiating wrapped objects that are used in an

interpreted system in which the interpreter is separate from the script. As noted several times in the prior responses above, this is a non-trivial difference requiring a completely different system architecture and methodology including: 1) creating an internal object; 2) subclassing objects; 3) dynamically instantiating objects upon demand; 4) recording object identifications in the script; and 5) other techniques mentioned earlier not required or anticipated by the cited Gurewich art. Given these differences, Applicants submit that Visual Basic does not anticipate the architecture of Applicants' invention.

Examiner's Comments: **(claims 20 and 47 continued)**

b. *means for integrating objects* (see at least chapter 20, section "The visual implementation of the WEST program", p. 581 et seq., see "Note", p. 584);

Applicants' Response:

Applicants incorporate and restate here their descriptions of the Visual Basic system and the system of Applicants' invention. The test which the Examiner references on page 585 says" "...writing the code of the formload() procedure." It is unnecessary to write any code in Applicants' invention to create a similar output. Applicants' scripts are not based on "computer code" that is compiled to p-code or machine code, rather the scripts are text strings that are parsed at runtime and interpreted. The means for integrating controls taught by Gurewich is completely different than the means used by Applicants' invention and, as such, can not be said to make obvious the novel and unique approach used by Applicants' in their invention.

Examiner's Comments: **(claims 20 and 47 continued)**

c. *means for sequencing objects* (see at least chapter 20, "Writing the code that performs the animation show", p. 594 et seq.); *and*

Applicants' Response:

The way Visual Basic sequences objects is based on the forms paradigm. A study of Visual Basic forms teaches that a form is a kind of window for controls, and that these controls are controlled by an executing program. In Visual Basic, when one such form is called by the interpreter and the resulting window becomes visible, all the controls within that form are instantiated. Controlling the appearance of these controls is a constant challenge to those familiar with the art of Visual Basic programming since all the controls within the form are active at once. An additional characteristic of the Visual Basic approach is that the controls consume memory at a great rate, and, as long as the form is open, all the memory needed by every control is taken up. When the form is shut, all the controls are de-instantiated, and the Visual Basic system is supposed to clear the memory. This happens with varying degrees of success so memory holes and leaks develop over time.

In contrast Applicants' invention provides total control over when the dynamically linked object is instantiated, and when it is de-instantiated (or killed). The objects are instantiated only upon demand and in accordance with the intention of the programmer as set forth in the script. This is done by the runtime system which reads and interprets a text script line by line. The system loads the standard VBX control into an instance of an intermediate (template) object that is internal to Applicants' invention and is used to wrap the standard VBX control as the run time interpreter sends data from the script to the standard control.

Examiner's Comments: **(claims 20 and 47 continued)**

d. *means for providing communication among objects wherein the functionalities performed by the system during execution are determined by the objects used and the script* (see at least chapter 20, "Writing the code of the FormLoad() procedure", p. 585 et seq.)

Applicants' Response:

For the reasons stated many times above, Gurewich, page 585, does not describe a system that stores the script separately and independently from the runtime system since the runtime system in Visual Basic is tied to the executable file through the DLL. Gurewich does not describe a system that uses a script. Gurewich does describe the use of Visual Basic software components in an environment with logical operators built into the Visual Basic program. As described many times above, Applicants' invention teaches the use of objects in a true object oriented system. The system Gurewich describes uses components as display items, and these display items, (buttons, dialogues, images, etc) are like ornaments to the program which is controlling the objects.

In Applicants' invention the objects are doing all the work. All the functions of the program are performed by objects, not by the interpreter; there are logical operator objects, there are arithmetic operator objects, there are data storage objects, there are display objects, and so on. Applicants' system does nothing without these objects. Contrast this with Visual Basic which can add, subtract, store data, and text values all without any objects. Thus, Visual Basic can perform functionalities without objects, but Applicants' invention can not. The unique use of objects to do all the work within Applicants' invention is not made obvious by

the use of such internal functions in Visual Basic.

Examiner's Comments: **Claims 21 and 48**

Gurewich discloses at least:

a. *means for instantiating objects* (see at least chapter 20, section "The visual implementation of the WEST program, p. 581 et seq.);

b. *means for integrating objects* (see at least chapter 20, section "The visual implementation of the WEST program, p. 581 et seq., see "Note", p. 584)

c. *means for sequencing objects* (see at least chapter 20, "Writing the code that performs the animation show", p. 594 et seq.); *and*

d. *means for providing communication among objects; wherein the functionalities performed by the system during execution are determined by the objects used and the script* (see at least chapter 20, "Writing the code of the FormLoad() procedure', p. 585 et seq.).

Applicants' Response:

Applicants incorporate their response with respect to claims 20 and 47 above. The Examiner has evaluated the steps within the claims and apparently believes that they seem to match up with information provided by Gurewich, but Applicants respectfully submit that the Examiner has not considered the preamble of the claim that precedes those steps and provides a basis for them, namely:

> *"A computer implemented run time system employing objects utilizing a minimum set of core functionalities which interprets a script, in which both the objects and the script may be maintained separately,"*

Visual Basic does not maintain the script separately and independently from the runtime system and objects as detailed above, and Visual Basic is not an object oriented software system. The Examiner's comparison is not valid. The only similarity found between Visual Basic and Applicants' invention lies in the fact that both use VBX controls. Each system uses these components very differently. Applicants' invention treats VBX controls like objects; Visual Basic does not. Applicants' invention uses VBX controls for logical operators, arithmetic operators, and system functions; Visual Basic does not. Applicants' invention creates instances of individual objects upon demand and within the control of users can deinstantiate individual objects not being used. Visual Basic can not do this. Visual Basic instantiates and maintains active until the form is closed all controls within a form window. Applicants' invention integrates objects by wrapping them in an intermediate object; Visual Basic does not. Visual Basic uses an underlying program written in Visual Basic code syntax. The Visual Basic program code is compiled to p-code. In Visual Basic these objects are then all instantiated at the same time within a form window. In contrast, Applicants' invention uses a text based script that is interpreted on a line by line basis, requiring the dynamic linking and use of VBX controls which have been subclassed within an internal (template) object. Applicants' invention sequences and instantiates object on a one by one basis. Therefore, Applicants' means of instantiating, integrating, and sequencing the objects is very different from the way Visual Basic uses VBX controls. Use of these components in the manner employed by Visual Basic can not and does not anticipate Applicants' invention.

Examiner's Comments: **Claims 22 and 49**

Gurewich discloses at least:

a. *a means for setting the values of properties and connecting events* (see at least chapter 14, p. 460 et seq.);

Applicants' Response:

Applicants respectfully submit that once again the Examiner has found a piece of information that might be related to object oriented programming if one is willing to omit from consideration the introductory phrase of the claim:

> *"A computer implemented system for employing objects, having property values and event connections, which can be set in time and turned on or off of a visually perceptible display device comprising:"*

At the time of Applicants' invention, the art of object oriented programming did not teach the setting of properties, which can be set in time and turned on and off, within a time line based system as described in Applicants' invention (see for example Fig. 3). Neither did the art described in Visual Basic teach this. Applicants respectfully submit that the Examiner's reliance on the use of VBX controls by both Visual Basic and Applicants' invention to suggest that Visual Basic anticipates Applicants' invention ignores the substantial differences in the manner of use of the controls by Visual Basic and Applicants' invention.

The use of a timeline user interface in an object oriented development system was unique at the time of Applicants' invention and was not taught by Visual Basic. Applicants' invention of a multi-tracked time line interface for arranging the dynamic instantiation of controls within an internal object in time is not taught by a system like Visual Basic that,

through a text based interface, treats VBX controls as components within the compiled code.

Examiner's Comments: **(claims 22 and 49 continued)**

b. *means for recording and maintaining a history of a plurality of properties settings and events connection as the settings and connections are changed* (see at least chapter 14, Table 14.1; chapter 15, table 15.1; chapter 19, table 19.1; chapter 20, table 20.1; and related discussion in the specification); *and*

Applicants Response:

The tables mentioned in Gurewich do list the objects and the settings of their respective properties. However, such tables are not found within the Visual Basic program and are only documentation conventions for the programmer. A more accurate representation of the way Visual Basic represents this information during coding of a Visual Basic project is represented on pages 462, 490, 565, and 586. As a documentation convention, the tables in chapters 14, 15, 19, and 20 are written on paper and are not utilized in program execution, and do not change automatically *"as the settings and connections are changed" (Applicants' claim)*. Clearly, such a presentation of the information on paper does not make obvious a dynamically created interactive scripting system that automatically records settings as the developer makes changes. It does not make obvious Applicants' system that transverses such a history on a change by change basis reinstating said changes in the development environment upon the command of the developer.

In addition, Applicants point out that the written form found in Gurewich does not show any history, and only represents the current state of the controls. In contrast, Applicants'

invention maintains in memory a step by step record (history) of each change made to the script as each change is made so that at any time the developer can step back through that history and recall a previous state. The history is not only recalled, but the state is re-established so each modification can be recalled and "undone", whether it is a modification to a property, event, @function, time track setting, or any other modification to the script. A program such as Visual Basic 3.0 that does not record settings as they are changed and does not maintain a history of modifications, can not and does not anticipate Applicants' invention.

Examiner's Comments: **(claims 22 and 49 continued)**

c. *means for traversing the history one change at a time wherein the property values and event connections may be edited from any point in the history* (see at least chapter 14, Table 14.1; chapter 15, Table 15.1; chapter 19; Table 19.1; chapter 20, Table 20.1; and related discussion in the specification).

Applicants' Response:

Applicants incorporate and restate here their answer to the Examiner's comments set forth immediately above in relation to subsection "b."

Examiner's Comments: **Claims 23 and 50**

Gurewich discloses at least:

a. *means for wrapping objects with additional properties beyond those properties and events internal to the objects* (see at least Gurewich, chapter 19, section "Attaching Code to the Multitasking Check box, page 573-576; chapter 20, section "The WEST Program" page

580 et seq., in particular the "Note" at page 584)

Applicants' Response:

Applicants incorporate their response with respect to claims 1, 28, 55, 56 and 57 above.

Examiner's Comments: **(claims 23 and 50 continued)**

b. *means for utilizing the additional properties and events to link and sequence the objects* (see at least chapter 20, section "The WEST Program" page 580 et seq., in particular the "Note" at page 584); *and*

Applicants' Response:

Applicants incorporate their response with respect to claims 1, 28, 55, 56 and 57 above. By way of further answer, Applicants note that since the cited reference does not teach anything about "additional properties and events," it is impossible for that same reference to teach how they are used.

Examiner's Comments: **(claims 23 and 50 continued)**

c. *means for reading one or more sets of property values and event settings maintained separately from the run time system and the objects wherein the execution of the objects is determined by the property values and event settings in the script* (see at least chapter 20, table 20.1 and related discussion in the specification)

Applicants' Response:

Visual Basic does not use a script, much less one that is maintained separately as

recited by the claim. As discussed many times above, Visual Basic creates compiled executable programs that link to a DLL holding the p-code interpreter. Gurewich, chapter 20, teaches how to write those programs in code. Since Applicants' claim addresses the way the runtime interpreter operates and uses the separately maintained script, any reference to Visual Basic only shows that the art of programming at the time of Applicants' invention had not anticipated such invention. Applicants' invention reads a script file into the runtime system, decompresses the text based script file, then starts to parse individual lines and begins interpretation. Each line in the script represents the information to be used in a single dynamically instantiated object. That object is instantiated and the results of its actions determine which line of the script is parsed, and subsequently which object is instantiated next. This is vastly different from the forms based system used in Visual Basic.

Applicants' system script does not link to a DLL. Applicants' system dynamically instantiates objects. These two features at the very minimum serve to distinguish Applicants' invention from Visual Basic and make it impossible for Visual Basic to anticipate Applicants' invention.

Examiner's Comments: **Claims 24 and 51**

The rejection of the base claim is incorporated. Gurewich further discloses *means for adding programming constructs or sub-languages utilizing objects* (see at least "Writing the code of the FormLoad() procedure", p. 585 et seq.; "Attaching code to the cmdPlay button', p. 591 et. seq).

Applicants' Response:

Applicants submit that they have overcome all bases for rejection of the base claim and that this dependent claim can not be obvious when the base claim is not obvious. By way of further answer, the use of objects as programming constructs is not anticipated by Visual Basic because Visual Basic has its own internal programming constructs. The addition of new programming constructs is a novel feature of Applicants' invention not taught by Visual Basic. In fact, the use of objects as programming constructs by Applicants' invention is unique to the point that even now, 10 years after the first development of Applicants' invention, other than the ones that ship with Applicants' invention, there are no objects that act as programming constructs to be found on the market.

The use of a subroutine written in Visual Basic to a Visual Basic program neither can nor does make obvious the unique ability of Applicant's invention to use a subclassed control that can operate as a programming construct, nor does it make obvious the addition of subclassed controls that contain whole programming languages and enable execution of programs developed in totally different programming languages like JAVA or a scripting system like Macromedia Flash. Incorporating different languages into one system is non-trivial and at the time of Applicants' invention, novel.

Examiner's Comments: **Claims 25 and 52**

Gurewich discloses at least:

a. *means for wrapping objects with additional properties and events beyond those properties and events internal to the object* (see at least Gurewich, chapter 19, section

"Attaching Code to the Multitasking Check Box, page 573-576; chapter 20, section "The WEST Program" page 580 et seq., in particular the "Note" at Page 584);*and*

Applicants' Response:

Applicants incorporate their response with respect to claims 1, 28, 55, 56 and 57 above.

Examiner's Comments:

b. *means for utilizing the additional properties and events to link and sequence the objects wherein the run time executions of the objects is determined by the property values and events* (see at least chapter 20, sections "The WEST Program" page 580 et seq., in particular the "Note" at page 584)

Applicants Response:

Applicants incorporate their response with respect to claims 1, 28, 55, 56 and 57 above. Note particularly the sections about wrapping to add properties and events and how line by line interpretation of the script works with dynamic object instantiation based on the actions of other objects.

Additionally, Applicants respectfully submit that in evaluating this claim the Examiner seems to have overlooked the preamble, which provides context for the body of the claim:

> *"A computer implemented system which interprets a script containing property values and event settings, which may be maintained separately[1], that distributes[2] processing to objects, provides and manages data flow among objects, and manages the execution[3] of objects comprising:"* [superscripts added for reference to the following]

Visual Basic does not make Applicants' invention obvious for the following reasons:

[1)] Visual Basic does not use a script, and it does not maintain the compiled p-code separately.

[2)] Visual Basic does not distribute processing to object oriented programming objects since it does not use such objects but rather uses VBX controls as components.

[3)] Visual Basic instantiates all objects within a form window when that window is opened. Visual Basic relies on the code in that window to act upon the controls which reside therein. Applicants' invention instantiates individual objects as required by the script and the actions of the objects. A system such as Visual Basic can not be said to anticipate the dynamic management of the execution of objects taught and enabled by Applicants' invention.

Since Visual Basic is so dissimilar to Applicants' invention, Applicants respectfully submit that the Examiner is in error in basing his rejection on the belief that Visual Basic anticipates Applicants' invention. Further, it should be noted that the execution of components in Visual Basic is determined by the program code since that is where the control structures reside, and as a result the VBX controls obey commands in the program code. In contrast, in Applicants' invention, the objects themselves are the control structures so it is the objects that determine the program flow. The objects use the information stored in the script to set their initial values, and they rely on the script to tell the runtime engine how to move data about, but ultimately it is the objects that control the actions of the application.

Examiner's Comments: **Claims 26 and 53**

Gurewich discloses at least:

a. *means for wrapping objects with additional properties and events beyond those properties and events provided internal to the object* (see at least Gurewich, chapter 19, section "Attaching Code to the Multitasking Check Box, page 573-576; chapter 20, section "The WEST Program" page 580 et seq., in particular the "Note" at page 584);

Applicants' Response:

Applicants incorporate their response with respect to claims 1, 28, 55, 56 and 57 above.

Examiner's Comments: **(Claims 26 and 53 continued)**

b. *means for utilizing the additional properties and events to link and sequence the objects*(see at least chapter 20, section "The WEST Program" page 580 et seq., in particular the "Note" at page 584); *and*

Applicants' Response:

Applicants incorporate their response with respect to claims 1, 28, 55, 56 and 57 above.

Examiner's Comments: **(Claims 26 and 53 continued)**

c. *means for specifying the temporal relationship among objects by placing the objects on one or more time lines wherein execution of the objects occurs at least partially concurrently and during which property values may be exchanged among the objects and events may be initiated* (see at least chapter 20, table 20.1, "cmdSlow, cmdFast, cmNormal and related discussion in the specification).

Applicants Response:

The table 20.1 in Chapter 20 does not represent the temporal relationship among objects on a time line. The tabular form of documentation is in no way similar to a timeline, as shown in Figure 3 of the present application document. The reference cited by the Examiner discloses [page 586, line 31, (Sub Form_Load ()] that one of the VBX controls contains an internal property that controls the tempo of multimedia playback.

The internal timing property referred to in Gurewich relates to the speed of playback, not the timing of instantiation. Unlike the reference cited by the Examiner, the time line found in Applicants' invention organizes the timing of instantiation and can be used to organize any object within the system since properties (including timing) are added by the wrapper and become common to all the objects once loaded into the system. Applicants respectfully submit that one internal property, speed of playback, of one VBX control as cited by the Examiner is not sufficient to make obvious a general solution of a time line interface upon which all object oriented programming objects can be placed in Applicants' invention.

The example cited by the Examiner further describes how the media object has other properties (repeat, pause, stop) which can be utilized by the WEST program. This shows an example of how to use the internal properties of VBX controls, and shows the broad utility of component based software. It does not anticipate the creation of a user interface for multimedia development that uses standard objects on a time line, or series of time lines as found in Applicants' invention.

Examiner's Comments: **Claims 27 and 54**

Gurewich discloses at least:

a. *means for wrapping objects with additional properties and events beyond those properties and events provided internal to the object* (See at least Gurewich, chapter 19, section "Attaching Code to the Multitasking Check Box, page 573-576; chapter 20, section "The WEST Program" page 580 et seq., in particular the "Note" at page 584);

Applicants' Response:

Applicants incorporate their response with respect to claims 1, 28, 55, 56 and 57 above.

Examiner's Comments: **(Claims 27 and 54 continued)**

b. *means for utilizing the additional properties and events to link and sequence the objects*(see at least chapter 20, section "The WEST Program" page 580 et seq., in particular the "Note" at page 584); and

Applicants' Response:

Applicants incorporate their response with respect to claims 1, 28, 55, 56 and 57 above.

Examiner's Comments: **(Claims 27 and 54 continued)**

c. *means for specifying a list of property values and event settings wherein the execution of the objects is determined by the list property values and event settings* (See at least chapter 19, Table 19.1 and related discussion in the specification).

Applicants' Response:

Claims 27 and 54 are based on the use of objects as programming constructs, which, as

shown above, Visual Basic does not do. Accordingly, Visual Basic can not anticipate Applicants' invention. The means used in Applicants' invention to specify the property list and event settings are through the use of an object browser, and not through a code based programming language as found in Visual Basic, C++, or ADA. Because Applicants' invention uses the unique "@" functions mentioned above, it is this record of data movements that determines the runtime contents of properties at execution. This facility provides the unique ability for developers to create whole applications with the mouse and without ever using the keyboard to enter syntactical code to command objects to perform functions ordinarily accomplished by the creation of program code that is later compiled and executed. Once recorded in the script, these settings are used by the program's interpreter to instantiate objects according to the instructions of the developer and to control the execution of the objects as intended. For all the foregoing reasons, Visual Basic does not anticipate Applicants' invention.

Examiner's Paragraph 7:

The Examiner has rejected claims 2-3 and 29-30 under 35 U.S.C. 102(b) as being anticipated by U.S. Patent No. 5,187,788 to Marmelstein, February 16, 1993.

Examiner's Comments: **Claims 2 and 29**

Marmelstein discloses at least:

a. *means for simultaneously displaying a plurality of different representations of the program structure* (see at least Figure 7 and related discussion in the specification); and

b. *means for manipulating the program structure within each of the different representations wherein the representations of the program structure may be synchronized* (see at least Figure 7 and related discussion in the specification).

Applicants' Response:

Marmelstein teaches the use of a flowchart based software system to generate ADA source code. The first line of the abstract to the patent states "(APEX) is an automatic code generation tool for the ADA language". Applicants have neither disclosed nor claimed to have invented such a thing as this because Applicants' invention is not a source code generator. The preamble of Applicants' claim 2 states that the invention is for generating an application script. As explained earlier: 1) the script is interpreted by a runtime program and is not an executable program; and 2) there is a significant difference between ADA source code that is later compiled into executable machine code and a script that is interpreted. One salient aspect and consequence of this difference is that in Applicants' invention it is possible to execute the developed application within the development environment and trace the execution by watching the icons highlight as they become instantiated. This is not possible with Marmelstein's invention because execution of the resulting ADA machine code executable occurs outside the scope of Marmelstein's invention.

Marmelstein clearly states in the Abstract that each of the interfaces he calls "views" is used for one specialized task. Most importantly, Marmelstein's system is designed so the interfaces would be used in sequence. The patent teaches: "The first view allows the programmer to lay out his initial ADA package specification." The form this interface takes is a flowchart. "The second view allows the programmer to create and manipulate complex data

structures." The form this interface takes is a series of dialogue boxes as are commonly used in the programmer's art. "The last view allows the programmer to define the control flow of his subprogram". The form this interface takes is a flowchart, like the first view.

In contrast to Marmelstein's approach, Applicants' invention can be used in such a way as to build and see a complete application in any one of Applicants' four views. Applicants' invention provides views that are useful according to their form, but the views are neither as specialized as Marmelstein's nor must they be used sequentially and for that reason each view can show the complete structure of the program. While Marmelstein claims that the views show the program structure, what each view actually does is to show just part of the structure of the program according to its function. Applicants' invention can show the complete structure of the program from any view, and for this reason is not made obvious by Marmelstein's teaching.

One of the views in Applicants' invention is a flowchart view that provides much of the capability of Marmelstein's first and third views combined. The second view in Marmelstein's invention is like the object browser in Applicants' invention (not considered a separate "view" by Applicants - not one of the four views taught by Applicants) with an important difference. Marmelstein's invention is used to create subprograms written in the ADA language, so it needs to allow the programmer to create data structures; whereas Applicants' invention uses existing unchangeable VBX controls so the object browser in Applicants' invention can only be used to manipulate the data within the VBX controls and the wrapper. Marmelstein has no view that is analogous to the Output View in Applicants' invention which shows the User Interface that is presented to the end user, nor does Marmelstein have a view that is equivalent

to the time line view in Applicant's invention.

Characterization of equivalent aspects of Marmelstein's method with terms used in Applicants' invention indicates that Marmelstein's method employs two separate and complementary flowchart views and an object browser that can be used to create new objects. This is in contrast to Applicants' system which has four distinct views and an object browser and utilizes preexisting components.

Applicants respectfully submit that the Examiner should withdraw his rejection based on Marmelstein for the following reasons:

1. Applicants' invention is not a code generator. As stated in the preamble of Claim 2, Applicants' invention generates a script that is subsequently interpreted by a runtime executable. Marmelstein's invention generates source code that is compiled into an executable program. Marmelstein's example clearly states that his system is a code generator for the ADA language.
2. Applicants' invention has four (4) views, each of which is fundamentally different from the other: time line, user interface, flowchart, and text grid. Marmelstein claims 3 views, but in equivalent terms presents only two flowchart views, each viewing a different aspect of the code, and a dialogue box for defining the objects. The integration of 4 distinctly different full views is non-trivial and not made obvious by a Marmelstein.
3. None of Marmelstein's views show the complete program structure but rather each shows part of the program structure according to its ability, whereas in Applicants' invention, each view can show the complete program structure and a

program can be completely created in any view.

Examiner's Comments: **Claims 3 and 30**

The rejection of the base claim is incorporated. Marmelstein Further discloses *a means for highlighting the icon for each object in the representations as objects are being instantiated during application development playback preview* (see at least Figure 7, the highlighted box "Deposit Display" and related discussion in the specification).

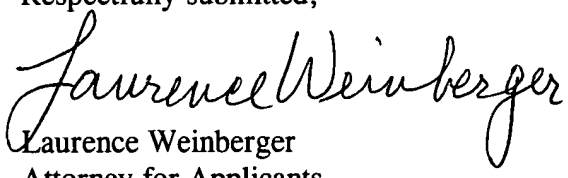
Applicants' Response:

Applicants have diligently searched the Marmelstein patent for the words "playback" and "preview". Neither word was found. Applicants also searched for and examined the occurrences of the word "development" to see if it was used in the context cited by the Examiner. Applicants could find no such instance. Similarly, Applicants examined the occurrence of the word "highlight" and found it only used in the editing and creation process. Applicants respectfully submit that the Examiner has misinterpreted the Marmelstein disclosure. The only Marmelstein teaching which relates to highlighting of an icon is the discussion related to the highlighting of the "Deposit Display" discussed in the Marmelstein specification (columns 24,25,26,27). Marmelstein explains that, when a window is selected during the editing or creation phase of development, the window is highlighted. This is standard user interface behavior. Claims (3 and 30) in the present application refer to the highlighting of icons and objects in each of the 4 views during execution from within the development environment during playback preview. Accordingly, Marmelstein does not anticipate Applicants invention.

Applicants submit that they have fully responded to all issues raised by the Examiner and respectfully request that the Examiner pass the application to allowance.

Dated: October 14, 2003

Respectfully submitted,



Laurence Weinberger
Attorney for Applicants
USPTO Reg. No. 27,965
Suite 103, 882 S. Matlack St.
West Chester, PA 19382
610-431-1703
610-431-4181 (fax)
larry@lawpatent.com